

# Two Flow-Based Approaches for the Static Relocation Problem in Carsharing Systems

Sahar Bsaybes\*, Alain Quilliot, Annegret K. Wagler, and Jan-Thierry Wegener\*

Université Blaise Pascal (Clermont-Ferrand II)  
Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes  
BP 10125, 63173 Aubière Cedex, France  
{bsaybes,quilliot,wagler,wegener}@isima.fr

**Abstract.** In a carsharing system, a fleet of cars is distributed at stations in an urban area, customers can take and return cars at any time and station. For operating such a system in a satisfactory way, the stations have to keep a good ratio between the numbers of free places and cars in each station. This leads to the problem of relocating cars between stations, which can be modeled within the framework of a metric task system. In this paper, we focus on the Static Relocation Problem, where the system has to be set into a certain state, outgoing from the current state. We present two approaches to solve this problem, a fast heuristic approach and an exact integer programming based method using flows in time-expanded networks, and provide some computational results.

## 1 Introduction

Carsharing is a modern way of car rental, attractive to customers who make only occasional use of a car on demand. In a carsharing system, a fleet of cars is distributed at specified stations in an urban area, customers can take a car at any time and station and return it at any time and station, provided that there is a car available at the start station and a free place at the final station. To ensure the latter, the stations have to keep a good ratio between the number of places and the number of cars in each station. This leads to the problem of balancing the load of the stations, called *Relocation Problem*: an operator has to monitor the load situations of the stations and to decide when and how to move cars from “overfull” stations to “underfull” ones.

Balancing problems of this type occur for any car- or bikesharing system, but the scale of the instances and the possibility to move one or more vehicles in balancing steps differ. We consider an innovative carsharing system, where the cars are partly autonomous, which allows to build wireless convoys of cars leaded by a special vehicle, such that the whole convoy is moved by only one driver (cf. [5]). This setting is similar to bikesharing, where trucks can simultaneously

---

\* This work was founded by the French National Research Agency, the European Commission (Feder funds) and the Région Auvergne within the LabEx IMobS3.

move several bikes during the relocation process [2,3,4]. The main goal is to guarantee a balanced system during working hours (dynamic situation as in [4,7]) or to set up an appropriate initial state for the morning (static situation as in [3]). Both, the dynamic and the static versions are known to be  $\mathcal{NP}$ -hard [1,3], a combinatorial approximation algorithm REOPT [6] and different heuristic approaches have been developed, see e.g. [7,8,9].

In this paper, we address the static situation where the system has to be set into a certain state, outgoing from the current state, within a given time horizon (Static Relocation Problem). In Section 2, we model the problem within the framework of a metric task system, where the tasks consist in moving cars out of overfull stations into underfull ones. Then, we present an exact and a heuristic approach to solve this problem. In order to obtain an exact solution, we interpret the Static Relocation Problem by means of flows in a time-expanded network (as e.g., proposed by [3] for bikesharing systems), see Section 3. Hereby, the two flows are not independent or share arc capacities as in the case of multicommodity flows, but are coupled in the sense that the flow of cars is dependent from the flow of drivers (since cars can only be moved in convoys). As obtaining an exact solution requires long computation times and the approximation algorithm from [6] does not always find a feasible solution, we propose a heuristic in Section 4, which firstly computes coupled flows in an aggregated network, and afterwards generates tours from these flows by time-expansion. We show that this approach yields optimal solutions under certain conditions and finally provide some computational results for both approaches in comparison with a lower bound and the approximation algorithm from [6], and close with some remarks and future lines of research.

## 2 Problem Description and Model

We model the Relocation Problem in the framework of a metric task system.

By [6], the studied carsharing system can be understood as a discrete event-based system, where the system components are the stations  $v_1, \dots, v_n$ , each having an individual capacity  $\text{cap}(v_i)$ , a system state  $\mathbf{z}^t \in \mathbb{Z}^n$  specifies for each station  $v_i$  the number of cars  $z_i^t$  at a time point  $t \leq T$  within a time horizon  $[0, T]$  and  $\mathbf{z}^t$  changes when customers or convoy drivers take or return cars at stations.

An operator monitors the evolution of system states over time and decides when and how to move cars from overfull stations to underfull ones, in order to avoid infeasible system states  $\mathbf{z}^t$  with  $z_i^t > \text{cap}(v_i)$  or  $z_i^t < 0$  for some station  $v_i$ . More precisely, a *task* is defined by  $\tau = (v_i, x)$  where  $x \in \mathbb{Z} \setminus \{0\}$  cars are to pickup (if  $x > 0$ ) or to deliver (if  $x < 0$ ) at station  $v_i$  within the time-horizon  $[0, T]$ . We call a task *oversatisfied* if more than  $|x|$  cars are picked up (resp. dropped) at  $v_i$ .

To fulfill these tasks, we create tours for the convoys in order to perform the desired relocation process. For that, it is suitable to encode the urban area where the carsharing system is running as a *metric space*  $M = (V, d)$  induced by a weighted graph  $G = (V, E)$  with weight function  $w : E \rightarrow \mathbb{R}_+$ , where the nodes

correspond to stations, edges to their physical links in the urban area, and the distance  $d$  between two points  $v_i, v_j \in V$  is the length of a shortest path from  $v_i$  to  $v_j$ . All drivers begin and end their work at the same location, the so-called depot. The depot is represented in  $V$  by a distinguished origin  $v_0 \in V$ .

This together yields a *metric task system*, a pair  $(M, \mathcal{T})$  where  $M = (V, d)$  is the above metric space and  $\mathcal{T}$  a set of tasks, as suitable framework to embed the tours for the convoys. A driver able to lead a convoy plays the role of a server, the number of available drivers is denoted by  $k$ . Each server has capacity  $L$ , corresponding to the maximum possible number of cars per convoy; several servers are necessary to serve a task  $\tau = (v, x)$  if  $x > L$  holds.

More precisely, we define the following. A *move* from one station to another is  $m = (j, v, t_v, v', t_{v'}, x)$ , where  $j \in \{1, \dots, k\}$  specifies the driver  $driv(m)$  that has to move from the origin station  $orig(m) = v \in V$  starting at time  $dep(m) = t_v$  to destination station  $dest(m) = v' \in V$  arriving at time  $arr(m) = t_{v'}$ , and a load of  $load(m) = x$  cars. Hereby, we require that

- $0 \leq load(m_i) \leq L$ ,
- from  $orig(m) \neq dest(m)$  follows  $arr(m) = dep(m) + d(orig(m), dest(m))$ .

We speak about a *waiting move* if  $orig(m) = dest(m)$  holds (which might be necessary if a proceeding move cannot be performed before a certain time point).

A *tour* is a sequence  $\Gamma = (m_1, m_2, \dots, m_n)$  of moves, starting and ending in the depot, with

- $driv(m_1) = \dots = driv(m_n)$ ,
- $dest(m_i) = orig(m_{i+1})$ ,
- $arr(m_i) = dep(m_{i+1})$ .

A move  $m_i$  *picks up* cars at  $orig(m_i)$ , if  $load(m_{i-1}) - load(m_i) < 0$ , and  $m_i$  *drops* cars at  $dest(m_i)$ , if  $load(m_i) - load(m_{i-1}) < 0$ . The *length of a tour* corresponds to the distance traveled by the driver. Several tours are composed to a transportation schedule. A collection of tours  $\{\Gamma_1, \dots, \Gamma_k\}$  is a *feasible transportation schedule*  $\mathcal{S}$  for  $(M, \mathcal{T})$  if

1. every driver has exactly one tour,
2. each task  $\tau \in \mathcal{T}$  is served (i.e., for every task  $\tau = (v, x)$ , the number of cars picked up (resp. dropped) at station  $v$  sum up to  $x$ ,
3. all system states  $\mathbf{z}^t$  are feasible during the whole time horizon  $[0, T]$ .

The *total tour length* of a transportation schedule is the sum of the lengths of its tours. Condition 3 requires that, besides the canonical precedences between a move  $m_i \in \Gamma$  and its successor move  $m_{i+1} \in \Gamma$ , also dependencies between tours are respected if *preemption* is used, i.e., if a car is transported in one convoy from its origin to an intermediate station, and from there by another convoy to its destination. This causes dependencies between tours, since some moves cannot be performed before others are done without leading to infeasible intermediate states (the reason why tours may contain waiting moves). More precisely, there is a precedence between move  $m_i \in \Gamma$  and move  $m'_j \in \Gamma'$  avoiding a system state  $\mathbf{z}^t$  with  $z_v^t < 0$  (resp.  $cap(v) < z_v^t$ ), if one of the following conditions is true:

- the move  $m_i$  drops cars at an overfull station,
- the move  $m_j$  picks up cars at underfull station.

We call a transportation schedule *non-preemptive* if there are no precedences between moves of different tours, and *preemptive* otherwise.

Our goal is to construct transportation schedules of minimal total tour length for the Relocation Problem in the static situation; hereby, all tours have to start and to end in the depot  $v_0$ , and preemption is allowed.

*Problem 1 (Static Relocation Problem  $(M, \mathbf{z}^0, \mathbf{z}^T, k, L)$ ).* Given a metric space  $M = (V, d)$  induced by a weighted graph  $G = (V, E, w)$ , start state  $\mathbf{z}^0 \in \mathbb{N}^{|V|}$ , destination state  $\mathbf{z}^T \in \mathbb{N}^{|V|}$  with  $|\mathbf{z}^0| = |\mathbf{z}^T|$  and time horizon  $T$ ,  $k$  servers of capacity  $L$ , find a transportation schedule of minimal total tour length for the metric task system  $(M, \mathcal{T})$  where  $\mathcal{T}$  consists of the tasks  $\tau = (v_i, z_i^0 - z_i^T)$  for all  $v_i$  with  $z_i^0 \neq z_i^T$ .

### 3 Min-Cost Flows in Time-Expanded Networks

In this section, we give an exact approach for the Static Relocation Problem  $(M, \mathbf{z}^0, \mathbf{z}^T, k, L)$  by defining a time-expanded network with two coupled flows. For that, we build a directed graph  $G^T = (V_T, A_T)$ , with  $A_T = A_H \cup A_L$ , as a time-expanded version of the original network  $G$  encoding the metric space. The cars and drivers will form two flows  $f$  and  $F$  through  $G^T$  which are coupled in the sense that on those arcs  $a \in A_L$  used for moves of convoys, we have the condition  $f(a) \leq L \cdot F(a)$  reflecting the dependencies between the two flows.

*Time-expanded network  $G^T = (V_T, A_T)$ .* The node set  $V_T$  is constructed as follows: for each station and the depot  $v \in V$  and each time point  $t$  in the given time horizon  $[0, T]$ , there is a node  $(v, t) \in V_T$  which represents station  $v$  at time  $t$ . The arc set  $A_T = A_H \cup A_L$  of  $G^T$  is composed of two subsets:

- $A_H$  contains, for each station  $v \in V$  of the original network and each  $t \in \{0, 1, \dots, T-1\}$ , the *holdover arc* connecting  $(v, t)$  to  $(v, t+1)$ .
- $A_L$  contains, for each edge  $(v, v')$  of  $G$  and each point in time  $t \in T$  such that  $t + d(v, v') \leq T$ , the *relocation arc* from  $(v, t)$  to  $(v', t + d(v, v'))$ .

Note, that the time-expanded network  $G^T$  is acyclic by construction.

*Flows in  $G^T$ .* On the time-expanded network  $G^T$ , we define two different flows, the car flow  $f$  and the driver flow  $F$ , and specify the capacities as well as the costs for each arc with respect to both flows.

A flow on a relocation arc corresponds to a move in a tour, i.e., some cars are moved by drivers in a convoy from station  $v$  to another station  $v'$ . Hereby, the stations can be used to pick up or to drop cars, or simply to transit a node (when a driver/convoy passes the station(s) on its way to another station). A relocation arc from  $(v, t)$  to  $(v', t + d(v, v'))$  has infinite capacity for the drivers. However, in order to ensure that cars are moved only by drivers and only in

convoys of capacity  $L$ , we require that  $f(a) \leq L \cdot F(a)$  for all  $a \in A_L$ . Thus, the capacities for  $f$  on the relocation arcs are not given by constants but by a function, see (1g). Each relocation arc  $a = ((v, t), (v', t + d(v, v')))$  corresponding to edge  $(v, v')$  has cost  $C(a) := d(v, v')$ .

A flow on a holdover arc corresponds to cars/drivers remaining at the station in the time interval  $[t, t + 1]$ . Therefore, the capacity of all holdover arcs with respect to flow  $f$  is set to  $\text{cap}(v)$ , see (1f), whereas there is no capacity constraint for  $F$  on such arcs.

To correctly initialize the system, we use the nodes  $(v, 0) \in V_T$  as sources for both flows and set their balances accordingly to the initial number of cars at station  $v$  and time 0 and locate the drivers at the depot  $v_0$ , see (1b). For all internal nodes  $(v, t) \in V_T$  with  $0 < t < T$ , we use normal flow conservation constraints, see (1d) and (1e). To ensure that the destination state is reached and each driver returns to the depot, we use as sinks the nodes  $(v, T)$ ,  $v \in V$ , for the car flow and the node  $(v_0, T)$  for the driver flow, and set their balances accordingly to  $z^T$  resp. to the number of drivers, see (1c).

The objective is to minimize the total tour length, see (1a).

$$\begin{aligned}
\min \quad & \sum_{a \in A_L} C(a)F(a) & (1a) \\
\sum_{a \in \delta^-(v, 0)} f(a) &= z_v^0, \quad \sum_{a \in \delta^-(v_0, 0)} F(a) = k & \forall (v, 0) \in V_T \quad (1b) \\
\sum_{a \in \delta^+(v, T)} f(a) &= z_v^T, \quad \sum_{a \in \delta^+(v_0, T)} F(a) = k & \forall (v, T) \in V_T \quad (1c) \\
\sum_{a \in \delta^-(v, t)} f(a) &= \sum_{a \in \delta^+(v, t)} f(a) & \forall (v, t) \in V_T, 0 < t < T \quad (1d) \\
\sum_{a \in \delta^-(v, t)} F(a) &= \sum_{a \in \delta^+(v, t)} F(a) & \forall (v, t) \in V_T, 0 < t < T \quad (1e) \\
0 \leq f(a) &\leq \text{cap}(v) & \forall a = [(v, t), (v, t + 1)] \in A_H \quad (1f) \\
f(a) &\leq L \cdot F(a) & \forall a \in A_L \quad (1g) \\
f, F &\text{ integer,} & (1h)
\end{aligned}$$

where  $\delta^-(v, t)$  denotes the set of outgoing arcs of  $(v, t)$ , and  $\delta^+(v, t)$  denotes the set of incoming arcs of  $(v, t)$ . Note, due to the flow coupling constraints (1g), the above constraint matrix is not totally unimodular (as in the case of uncoupled flows). This reflects that the problem is  $\mathcal{NP}$ -hard.

Finally, the flows in the time-expanded network have to be interpreted as a transportation schedule. Hereby, car and driver flows on relocation arcs clearly correspond to moves. The outcome is a preemptive transportation schedule which is feasible since all dependencies over time are properly respected by the flow conservation constraints. This implies:

**Theorem 1.** *The optimal solution of system (1) corresponds to a preemptive transportation schedule with minimal total tour length for the Static Relocation Problem  $(M, \mathbf{z}^0, \mathbf{z}^T, k, L)$ .*

## 4 Lifted Flows in Aggregated Networks

The computation times for computing an exact solution by solving the integer linear program (1a)–(1h) are extremely high even for small instances (see Table 1). This motivates the research for heuristics which compute a good feasible solution within a reasonable time. Therefore, we describe in this section a heuristic approach of lifted flows in aggregated networks (LIFTFLOW) to solve the Static Relocation Problem  $(G, \mathbf{z}^0, \mathbf{z}^T, k, L)$ , where  $G$  is the complete weighted graph  $G = (V_O \cup V_U \cup \{v_0\}, E, d)$  containing the overfull stations  $V_O$  (with  $z_i^0 > z_i^T$ ), the underfull stations  $V_U$  (with  $z_i^0 < z_i^T$ ), a depot  $v_0$ , all connections  $E$  between them and distances  $d: E \rightarrow \mathbb{N}$ .

The approach LIFTFLOW is performed in two steps. Firstly, we construct a weighted complete bipartite graph and find a flow that starts and ends in the depot, while passing overfull and underfull stations, minimizing the costs. Each arc carrying a flow corresponds to a move between two stations. Secondly, we compute possible precedences between moves and from that we construct tours (with preemption) for all convoys.

*First step: Flows in aggregated networks.* In the first step, we construct an aggregated network and solve a min-cost flow problem on this network. Arcs carrying positive flows then correspond to the movement of drivers and cars in the network. The aggregated network is a directed, weighted graph  $G^A = (V_A, A_A, w)$ , based on the graph  $G$ .

The node set  $V_A = V_S \cup V_O \cup V_U \cup V_D$  is composed by the following nodes:  $V_S$  and  $V_D$  contain the depot,  $V_O$  contains the set of overfull stations,  $V_U$  contains the set of underfull stations.

The arc set  $A_A = A_{SO} \cup A_{OO} \cup A_{OU} \cup A_{OU} \cup A_D$  is composed of several subsets:

- the set of *start arcs*  $A_{SO} = \{(v_0, v_o) : v_0 \in V_S, v_o \in V_O\}$  connecting the depot to the overfull stations,
- the set of *overfull arcs*  $A_{OO} = \{(v_o, v'_o), (v'_o, v_o) : v_o, v'_o \in V_O\}$  connecting all overfull stations,
- the set of *connection arcs*  $A_{OU} = \{(v_o, v_u), (v_u, v_o) : v_o \in V_O, v_u \in V_U\}$  connecting overfull and underfull stations,
- the set of *underfull arcs*  $A_U = \{(v_u, v'_u), (v'_u, v_u) : v_u, v'_u \in V_U\}$  connecting all underfull stations, and
- the set of *sink arcs*  $A_D = \{(v_u, v_0) : v_u \in V_U, v_0 \in V_D\}$ .

The set containing all overfull, connection and underfull arcs is denoted by  $A_R := A_{OO} \cup A_{OU} \cup A_U$ . For an arc  $a = (v, v') \in A_A$  the arc weights  $w(a) := d(v, v')$  correspond to the distances.

On the aggregated network  $G^A$ , we define two different flows, the car flow  $f$  and the driver flow  $F$ , and specify the capacities as well as the costs for each arc with respect to both flows. Flow on an overfull, connection or underfull arc corresponds to a move in a tour, i.e., some cars are moved by drivers in a convoy from station  $v$  to another station  $v'$ . In order to ensure that cars are moved only by drivers and that the convoy capacity  $L$  is not exceeded, we require that  $f(a) \leq L \cdot F(a)$  holds for all relocation arcs  $a \in A_R$ , see (2g).

To correctly initialize the system, we use the depot  $v_0 \in V_S$  as source for the driver flow, see (2d), and the nodes  $v \in V_O$  as sources for the car flow and set their balances accordingly to the numbers of cars that have to be picked up at station  $v$ , i.e.,  $b^+(v) := \max\{z_v^0 - z_v^T, 0\} \geq 0$ , see (2b). Equalities (2f) and (2e) are the flow conservation constraints. The nodes of underfull station  $v \in V_U$  are the destinations of the cars. So we set their balances accordingly to the number of cars that have to be dropped at station  $v$ , i.e.,  $b^-(v) := \min\{z_v^0 - z_v^T, 0\} \leq 0$ , see (2c). Finally, the sink  $v_0 \in V_D$  is the destination of the  $k$  drivers, see (2d). We consider a min-cost flow problem where we intend to balance all stations with minimal costs (2a).

$$\min \sum_{a \in A_A} w(a)F(a) \quad (2a)$$

$$\sum_{a \in \delta^-(v)} f(a) - \sum_{a \in \delta^+(v)} f(a) = b^+(v) \quad \text{for all } v \in V_O \quad (2b)$$

$$\sum_{a \in \delta^-(v)} f(a) - \sum_{a \in \delta^+(v)} f(a) = b^-(v) \quad \text{for all } v \in V_U \quad (2c)$$

$$\sum_{a \in \delta^-(v_0)} F(a) = \sum_{a \in \delta^+(v_0)} F(a) = k \quad v_0 \in V_S, v_0 \in V_D \quad (2d)$$

$$\sum_{a \in \delta^-(v)} F(a) - \sum_{a \in \delta^+(v)} F(a) = 0 \quad \text{for all } v \in V_O \cup V_U \quad (2e)$$

$$\sum_{a \in \delta^-(v)} f(a) - \sum_{a \in \delta^+(v)} f(a) = 0 \quad \text{for all } v \in V_O \cup V_U \quad (2f)$$

$$0 \leq f(a) \leq L \cdot F(a) \quad \text{for all } a \in A_R \quad (2g)$$

$$f, F \text{ integer}, \quad (2h)$$

Note, due to constraints (2g), the above constraint matrix is again not totally unimodular.

*Second step: Compute transportation schedule.* Next, we describe how to compute a transportation schedule from the solution of the first step.

For that, we first compute “pre-moves”, which have the origin and destination stations as well as the load of a move, but no times. With these pre-moves we compute “pre-tours” as sequences of “pre-moves”. Formally, a *pre-move*  $\tilde{m} = (v, v', x)$  is a 3-tuple, where  $v = \text{orig}(\tilde{m})$  is the origin station,  $v' = \text{dest}(\tilde{m})$  the destination station and  $x = \text{load}(\tilde{m})$  the load of the pre-move; a *pre-tour* is a sequence of pre-moves.

Let  $f^A$  be the car flow and  $F^A$  the driver flow computed by the integer linear program (2) in the first step. Then every arc  $a = (v, v') \in A_A$  with  $F^A(a) > 0$  corresponds to a pre-move  $\tilde{m}_a = (v, v', x)$ , with  $x \leq f^A(a)$ . All pre-moves have to be assigned to a pre-tour in a feasible order, i.e., the destination station of each pre-move has to be equal to the origin station of the successor pre-move. This can be done by searching for paths within the aggregated network. However, in general, there is not only a unique path within the aggregated network leading to several possible pre-tours. Note that the aggregated network is not cycle-free. Thus, there can be isolated cycles in the solution. For this paragraph, let us assume that there are no such isolated cycles; a strategy to detect and handle isolated cycles is presented in the next paragraph.

Since we consider the preemptive situation, it is possible that there are precedences between different tours. We define a precedence relation between pre-moves in an analog way to the definition of precedences between moves (see Section 2). For two different pre-tours  $\tilde{I}$  and  $\tilde{I}'$  there exists a potential precedence between two pre-moves  $\tilde{m}_i \in \tilde{I}$  and  $\tilde{m}'_j \in \tilde{I}'$  ( $\tilde{m}_i$  precedes  $\tilde{m}'_j$ ) if

- the destination station  $v_o$  of  $\tilde{m}_i$  is an overfull station,  $\tilde{m}_i$  drops cars at  $v_o$ , and  $\tilde{m}'_j$  picks up cars at  $v_o$ ; or
- the origin station  $v_u$  of  $\tilde{m}'_j$  is an underfull station,  $\tilde{m}'_j$  picks up cars at  $v_u$ , and  $\tilde{m}_i$  drops cars at  $v_u$ .

A pre-tour without precedences to another tour, can be directly transformed to a tour, by computing the departure and arrival times of each (pre-)move, and then assigning all moves to a driver. Otherwise, there is a preemptive situation which means that one convoy transports cars to a station and another convoy picks up these cars afterwards. In order to ensure that the cars are dropped before they are picked up, we possibly have to add additional waiting moves to the final tour. For that we construct a *precedence relation* between pre-moves.

In some rare cases, the precedence relation is not acyclic. In this situation, we add an additional constraint to the integer linear program (2a)–(2h), to receive a new solution. For that we add

$$\sum_{a \in A_A} w(a)F(a) > \sum_{a \in A_A} w(a)F^A(a)$$

to the constraints (2b)–(2h) and recompute the steps above. In fact, in our set of randomly generated test-instances, this case never occurred (see Table 1), and we do not expect them to occur often in practice.

Thus, let us consider an acyclic precedence relation containing precedences between different pre-tours. The arrival and departure times of the moves are derived from the distances between the origin and destination stations of the pre-moves. When the departure time of a move is computed from a pre-move, having a precedence relation to a pre-move in another pre-tour then we have to compute the arrival time of the preceding pre-move before, in order to be able to compute the waiting time. If several pre-moves  $\tilde{m}_1, \dots, \tilde{m}_\lambda$  precede a pre-move  $\tilde{m}$  then, in general,  $\tilde{m}$  does not need to wait for all preceding pre-moves but only until there are enough cars at  $orig(\tilde{m})$ . For that, we compute



a minimal set  $S \subseteq \{\tilde{m}_1, \dots, \tilde{m}_\lambda\}$  by solving a minimum matching problem on the complete bipartite graph  $(\{\tilde{m}_1, \dots, \tilde{m}_\lambda\} \cup \{\tilde{m}\}, E, w)$ , where the arc weight  $w(a)$  of  $a = (\tilde{m}_j, \tilde{m})$  corresponds to  $load(\tilde{m}_j)$ . Hereby, the sum of the weights of the selected arcs and the number of cars at  $orig(\tilde{m})$  must be at least  $load(\tilde{m})$ . The waiting time for  $\tilde{m}$  is then induced by the latest arrival time of all  $\tilde{m}_j \in S$ .

After all waiting times have been computed, we construct a transportation schedule from the (waiting) moves.

*Handling cycles in the lifted flows.* Finally, we describe how isolated cycles in the lifted flows can be handled. Let  $v_1 \dots v_\lambda v_1$  be an isolated cycle with  $F^A(v_1, v_2) = \dots = F^A(v_{\lambda-1}, v_\lambda) = F^A(v_\lambda, v_1)$ . Furthermore, let the cycle be ordered so that  $v_1$  is an overfull station and  $v_\lambda$  is an underfull station and so that  $f^A(v_\lambda, v_1) = 0$ . Note, the conservation constraints ensure that isolated cycles contain overfull and underfull stations. Furthermore, the existence of an arc from an underfull to an overfull station with  $f^A(v_\lambda, v_1) = 0$  is ensured by the minimality of the car flows. Since the cycle is isolated, it holds by definition  $F^A(v, v_j) = F^A(v_j, v) = 0$  for all  $v \in V_A \setminus \{v_1, \dots, v_\lambda\}$ .

Since  $v_1$  is an overfull station and  $v_\lambda$  an underfull station, we can consider the path  $v_1 \dots v_\lambda$  as a subsequence of pre-moves. Precedences within this subsequence are then handled as described in the previous paragraph.

Let  $\Gamma$  be a tour from the transportation schedule computed in Step 2, and let  $m$  be the last move from  $\Gamma$ . Since there are no cars transferred from  $v_\lambda$  to  $v_1$ , we can modify  $\Gamma$  by removing  $m$  and adding a move from  $orig(m)$  to  $v_1$ , as well as the moves induced from the path  $v_1 \dots v_\lambda$ . Finally, a move from  $v_\lambda$  to the depot  $dest(m)$  is added.

Since we can handle precedences between pre-moves and the case of isolated cycles in the two flows, we obtain:

**Theorem 2.** *The approach LIFTFLOW computes a feasible (possibly preemptive) transportation schedule for the Static Relocation Problem.*

*Optimal solutions and lower bounds.* Under certain conditions, the algorithm LIFTFLOW computes an optimal solution for the Static Relocation Problem or provides a lower bound for an optimal solution.

**Theorem 3.** *Let  $(G, z^0, z^T, k, L)$  be a Static Relocation Problem,  $f^A$  and  $F^A$  optimal flows in the aggregated network  $G^A$ , and  $\mathcal{S}$  be the resulting transportation schedule. If  $f^A$  and  $F^A$  satisfy that*

1. *there are no cycles in the precedence graph,*
2. *there are no isolated cycles in the flows,*
3. *the capacities of the stations are sufficiently large,*
4. *the time horizon is sufficiently large, and*
5. *there exists an optimal solution so that no balanced station is used as pre-emption station,*

*then  $\mathcal{S}$  is an optimal solution for  $(G, z^0, z^T, k, L)$ . If only condition 5 is fulfilled, then  $\sum_{a \in A_A} w(a)F^A(a)$  is a lower bound for the tour length of an optimal solution for  $(G, z^0, z^T, k, L)$ .*

## 5 Computational Results

Both, the heuristic approach LIFTFLOW and the exact approach, have been tested on randomly generated instances (with 20–80 over-/underfull stations, 50–150 stations in total, convoy capacities 5 and 10, and 10–30 drivers). The stations are randomly distributed on a plane and the distances between two closest stations (w.r.t. the Euclidean metric) are kept as rounded integers in the graph. Hereby, we ensure that the graph is connected. The time horizons are set to 100 and 150. Note that the size of these instances corresponds to small car- or bikesharing systems or to clusters of larger systems, as in [9]. The combinatorial part of the algorithm LIFTFLOW has been implemented in Java 1.6, Gurobi 5.6 is used for solving the integer linear program (short: ILP) in the first step of the algorithm. Gurobi 5.6 is also used to solve the ILPs for the exact approach. The tests have been run on a server with a total of 160 Intel Xeon E7-8870 cores, each clocked at 2.40GHz, and with 1 TB RAM. The number of threads for solving an ILP is restricted to 16, the implementation of LIFTFLOW is single-threaded.

In the heuristic approach LIFTFLOW, most of the computation time is used for solving the min-cost flows in the aggregated network. Hereby, we observed that after 20 minutes computation time for the first step, the objective function value rarely improved, see Table 1 showing the results of solving the ILP in the first step of LIFTFLOW after 20 minutes and 4 hours. That the results are better in some cases after 20 minutes than after 4 hours is due to a different number of isolated cycles. The computation time for computing the transportation schedule from the flows in the second step is 40 seconds for all tours. Due to the small impact, these run times are not listed in the table.

For the exact approach, the optimal solution could not be found for any test instance within the given time limit. Hereby, we state the best found solution values from a set of different optimization parameters on solving the ILPs. The time limits have been set to at least 4 hours (once we have let the computation running for nearly 16 days). In the first part of the algorithm LIFTFLOW, the optimal solution was found in 6 instances within the time limits. In eight (resp. seven) examples, LIFTFLOW computes tours exceeding the given time horizon leading to a non-feasible transportation schedule. However, most of the times, these transportation schedules can be modified by splitting the concerning tours into two (or more) tours so that they fit in the time horizon. This proposal works if there are enough drivers available.

## 6 Conclusion

In this paper, we considered the Static Relocation Problem  $(G, z^0, z^T, k, L)$ , where tours for  $k$  drivers have to be computed in a graph  $G$ , whereas the maximal length of the tours must fit into a given time horizon  $T$ . Hereby, we compute preemptive transportation schedules: a car can be transported in one convoy from its origin to an intermediate station, and from there by another convoy to its destination. In order to have an exact solution we construct a time-expanded network  $G^T$  from the original network  $G$  and compute two coupled flows (a car

**Table 1.** This table shows the computational results for several test instances of the algorithm LIFTFLOW (the time limit was set to 20 minutes and 4 hours) in comparison to the value found by solving the ILP (the time limit was set at least 4h). For every parameter set we created two different test instances ( $a$  and  $b$ ). In this table, the following parameters and results are shown: the name of the test instance  $a$  or  $b$  (1st column), the total amount of stations (2nd column) and the number of overfull and underfull stations (3rd column). Furthermore, it shows the considered time horizon  $T$ , the server capacity  $L$ , the number of drivers  $k$ , the total tour length LF found by LIFTFLOW (after 20 minutes and 4 hours, respectively), a lower bound LB (see Theorem 3 assuming condition 5 holds), the duality gap between LIFTFLOW (20min) and LB in percent, by the ILP solver after 4 hours and the duality gap between ILP and LB, and by an algorithm with an approximation factor REOPT (see [6] for details) and the median runtime  $t$  in seconds of several runs. Values marked with an asterisk (\*) contain a tour exceeding the time horizon, i.e., it is not a feasible transportation schedule; the ILP within LIFTFLOW has been solved optimally when values are marked with a plus (+), otherwise the lower bound is computed by the ILP solver; the cells marked with a hyphen (-) indicate that no solution was found within the time limit.

instance	stations	$\pm$ stations	T	L	k	LF (20min)	LF (4h)	LB	GAP%	ILP	GAP%	REOPT	t(s)
$a$	050	10/10	100	05	010	272*	272*	232 <sup>+</sup>	14.7	312	25.6	372	2
$b$	050	10/10	100	05	010	315*	315	238 <sup>+</sup>	24.4	374	36.4	-	-
$a$	050	10/10	100	10	010	255*	255*	221 <sup>+</sup>	13.3	287	23.0	328	3
$b$	050	10/10	100	10	010	254	254*	188 <sup>+</sup>	23.3	292	35.6	-	-
$a$	050	20/20	100	05	020	319	310*	285	8.7	404	28.0	438	85
$b$	050	20/20	100	05	020	378*	373	334 <sup>+</sup>	11.6	467	28.5	519	78
$a$	050	20/20	100	10	020	291	291	245	6.5	369	26.3	410	70
$b$	050	20/20	100	10	020	282	292	262 <sup>+</sup>	7.1	385	31.9	420	60
$a$	100	30/30	100	05	030	533	513	435	15.4	736	38.7	-	-
$b$	100	30/30	100	05	030	448	432	361	16.5	616	39.3	673	482
$a$	100	30/30	100	10	030	396*	412*	322	11.4	510	31.2	-	-
$b$	100	30/30	100	10	030	327*	327*	269	12.5	470	39.1	610	496
$a$	100	40/40	100	05	030	536*	539*	471	8.2	765	35.7	-	-
$b$	100	40/40	100	05	030	469	469	367	16.0	599	34.2	668	1391
$a$	100	40/40	100	10	030	434*	434	340	9.7	585	33.0	-	-
$b$	100	40/40	100	10	030	388	417	285	14.4	473	29.8	614	713
$a$	150	50/50	150	05	030	686	690	535	17.2	944	39.5	-	-
$b$	150	50/50	150	05	030	717	707	594	13.3	1008	39.2	-	-
$a$	150	50/50	150	10	030	474	476	378	8.6	675	35.6	-	-
$b$	150	50/50	150	10	030	533	534	415	13.5	812	43.1	-	-
average									13.3		33.7		

and a driver flow) on this network with an integer linear program. Due to the coupling constraints, the obtained constraint matrix is not totally unimodular (as in the case of uncoupled flows).

Due to the high computational times of the exact approach and the fact that algorithm REOPT, which computes non-preemptive transportation schedules, not always finds a feasible solution, we developed a heuristic approach to solve the Static Relocation Problem: the algorithm LIFTFLOW. The construction of the tours by LIFTFLOW is as follows. In the first step, two coupled flows on a graph are computed, a driver and a car flow. These flows serve as input for the second step, where firstly a set of pre-tours is constructed. Afterwards, precedence relations between pre-tours are computed, and from these precedences and the set of pre-tours, we finally compute a transportation schedule.

The heuristic approach LIFTFLOW solves the test instances faster and, computes already after 20 minutes transportation schedules with shorter total tour lengths than the exact approach using time-expanded networks with a time limit of 4 hours (see Table 1).

Under certain conditions, LIFTFLOW computes an optimal solution. Otherwise, we receive at least a lower bound (Theorem 3).

There are several practical and theoretical open questions according to the Static Relocation Problem. Currently, we minimize the total tour length. Applying the ideas of LIFTFLOW so that the makespan is minimized is one goal for the future. Another future goal is to improve the runtime of solving the min-cost flow in the aggregated network of the algorithm LIFTFLOW. This may be achieved by improving the lower bound from the dual solution. Usually, every driver used gives additional costs. Thus, it is desirable to know the minimal and/or maximal number of drivers needed in order to solve the Static Relocation Problem within the given time horizon. To the best of our knowledge this is still an open question. Due to the time horizon, it is possible that there does not exist a feasible solution for a given instance at all. Having feasibility conditions is useful in two directions: to save unnecessary computation time for an algorithm and to generate test instances which can give feasible solutions. Thus, finding feasibility conditions as well as lower bounds for the time horizon is another future goal.

## References

1. M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, editors. *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*. Elsevier Science B.V., Amsterdam, 1995.
2. M. Benchimol, P. Benchimol, B. Chappert, A. de la Taille, F. Laroche, F. Meunier, and L. Robinet. Balancing the stations of a self service bike hire system. *RAIRO - Operations Research*, 45:37–61, 0 2011.
3. D. Chemla, F. Meunier, and R. Wolfler Calvo. Bike sharing systems: Solving the static rebalancing problem. pages 120–146, 2013.
4. C. Contardo, C. Morency, and L-M. Rousseau. Balancing a dynamic public bike-sharing system. Technical Report 9, CIRRELT, 2012. <https://www.cirrelt.ca/DocumentsTravail/CIRRELT-2012-09.pdf>.
5. M. EL-Zaher, B. Dafflon, F. Gechter, and J.-M. Contet. Vehicle platoon control with multi-configuration ability. *Procedia Computer Science*, 9(0):1503–1512, 2012.
6. S. O. Krumke, A. Quilliot, A. K. Wagler, and J.-T. Wegener. Models and algorithms for carsharing systems and related problems. *Electronic Notes in Discrete Mathematics*, 44(0):201 – 206, 2013.
7. S. O. Krumke, A. Quilliot, A. K. Wagler, and J.-T. Wegener. Relocation in carsharing systems using flows in time-expanded networks. In *Experimental Algorithms*, volume 8504 of *LNCS*, pages 87–98. Springer, 2014.
8. M. Rainer-Harbach, P. Papazek, B. Hu, and G. R. Raidl. Balancing bicycle sharing systems: A variable neighborhood search approach. In *EvoCOP*, pages 121–132, 2013.
9. J. Schuijbroek, R. Hampshire, and W.-J. van Hoes. Inventory rebalancing and vehicle routing in bike sharing systems. 2013. working paper.